ΠΤΥΧΙΑΚΉ ΕΡΓΑΣΊΑ

ΔΙΑΔΙΚΤΥΑΚΉ ΕΦΑΡΜΟΓΉ ΗΛΕΚΤΡΟΝΙΚΟΎ ΕΜΠΟΡΊΟΥ

ΟΝΟΜΑΤΕΠΩΝΥΜΟ ΦΟΙΤΗΤΩΝ: Ευσταθία Στραβοκεφάλου ΑΜ: 0860
Αναστασία Φωτακάκη ΑΜ: 0891

ΕΠΙΒΛΕΠΟΝΤΕΣ: Σωτήρης Χριστοδούλου
Παναγιώτης Αλεφραγκής

ΑΝΤΙΡΡΙΟ 2014

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή

Αντίρριο, 2014

ΕΠΙΤΡΟΠΉ ΑΞΙΟΛΌΓΗΣΗΣ

1 ,Υπογραφή

2 ,Υπογραφή

3 ,Υπογραφή

Περίληψη

Το ηλεκτρονικό εμπόριο στην εποχή μας βρίσκεται στο απόγειό του και ακολουθεί μια πορεία δύο ταχυτήτων. Τα μεγάλα διαδικτυακά καταστήματα της προηγούμενης δεκαετίας έχουν γίνει ακόμα πιο μεγάλα, πιο ασφαλή, αξιόπιστα και επιτρέπουν στους προγραμματιστές να κατασκευάζουν εφαρμογές που αλληλεπιδρούν με χρήστες και προϊόντα μέσω APIs που διαθέτουν. Τα μικρά καταστήματα θεωρούν την διαδικτυακή παρουσία εξίσου σημαντική με την φυσική. Υπάρχει μεγάλος αριθμός καταστημάτων με φυσική ή όχι παρουσία που προσπαθεί να αποκτήσει μερίδιο από την πίτα των ηλεκτρονικών πωλήσεων. Αυτήν την χαοτική κατάσταση επιλογών που έχει ένας επίδοξος αγοραστής προσπαθούν να λύσουν Πράκτορες Σύγκρισης Αγορών (Shoping Comparison Agents).

Abstract

Nowadays the ecommerce businesses landscape is at its peak but it seems is more like a two lane highway. In the fast lane there is a handful of colossal ecommerce businesses that during the last decade have scale enormously in terms of size, services and have been established in people's minds as trustworthy and secure. On the other hand in the slow lane, there are innumerable small businesses with or without physical presence that struggle to take a piece of the internet sales pie. People have difficulty searching in all those eshops which differ themselves in terms of trust shipping costs and prices. A solution to this chaotic situation is given by Shopping Comparison Agents.

Περιεχόμενα

Κατάλογος πινάκων

Κατάλογος σχημάτων

Κεφάλαιο 1

Εισαγωγή

Το ηλεκτρονικό εμπόριο στην εποχή μας βρίσκεται στο απόγειό του. Η διαδικτυακή παρουσία μικρών καταστημάτων γίνεται όλο και πιο συχνή και ο κόσμος πλέον θεωρεί τις ηλεκτρονικές αγορές αξιόπιστες. Αλλά μπορούμε να διακρίνουμε ότι η πρόοδος του ηλεκτρονικού εμπορίου ακολουθεί μια πορεία δύο ταχυτήτων.

Μπροστά βρίσκονται τα μεγάλα διαδικτυακά καταστήματα όπως το Amazon και το ebay. Με συνεχή εξέλιξη ετών, αυτά τα καταστήματα έχουν καθιερωθεί στις συνειδήσεις του κόσμου ως αξιόπιστα και ασφαλή. Επίσης η γκάμα προϊόντων που διαθέτουν είναι τεράστια, οπότε σε αυτό το πεδίο είναι εκτός συναγωνισμού από τη μεγαλύτερη πλειοψηφία των υπόλοιπων καταστημάτων.

Παρόλα αυτά μεγάλο πλήθος μικρότερων φυσικών καταστημάτων θεωρούν απαραίτητη τη διαδικτυακή παρουσία. Επίσης πλήθος αποκλειστικά διαδικτυακών καταστημάτων έχει δημιουργηθεί που επικεντρώνεται σε κυρίως τοπικές αγορές π.χ. επιπέδου χώρας.

Όλοι αυτοί συναγωνίζονται για ένα μερίδιο από την πίτα των ηλεκτρονικών αγορών. Είναι προφανές ότι επικρατεί μια χαοτική κατάσταση επιλογών για τον υποψήφιο αγοραστή. Έχει να επιλέξει μέσα από ένα μεγάλο πλήθος καταστημάτων με διαφορετικά περιβάλλοντα διεπαφής, διαφορετικούς τρόπους αποστολής, διαφορετικές χρεώσεις και αμφιβόλου αξιοπιστίας.

Την κατάσταση αυτή προσπαθούν να διαλευκάνουν Πράκτορες Σύγκρισης Αγορών (Shoping Comparison Agents) Αυτές οι εφαρμογές αναλαμβάνουν να καταγράφουν τα προϊόντα εκατοντάδων καταστημάτων, μαζί με δυνατές επιλογές αποστολής ή χρέωσης που έχουν. Επίσης επισκέπτονται ανά τακτά χρονικά διαστήματα τα καταστήματα για να καταγράψουν πιθανές μεταβολές στην κατάσταση των προϊόντων. Εκτός από την εξελιγμένη αναζήτηση που διαθέτουν για την εύρεση των πιο οικονομικών προϊόντων, μια εξίσου σημαντική λειτουργία που προσφέρουν είναι ότι βαθμολογούν τα καταστήματα ως προς την αξιοπιστία τους. Αυτό

είναι πολύ σημαντικό και για τα καταστήματα και για τους αγοραστές γιατί αν ένας αγοραστής δεν θεωρεί το κατάστημα αξιόπιστο δεν αγοράζει από αυτό.

Στην Ελλάδα η μεγαλύτερη διαδικτυακή εφαρμογή πράκτορες σύγκρισης αγορών είναι το skroutz.gr. Ειναι φτιαγμένο με το εργαλείο κατασκευής διαδικτυακών εφαρμογών Ruby on Rails. Επίσης διαθέτει API που επιτρέπει στους προγραμματιστές να αποκτήσουν μερική πρόσβαση στη βάση δεδομένων προϊόντων και καταστημάτων που διαθέτει.

Στο δεύτερο κεφάλαιο της εργασίας αναλύουμε και συγκρίνουμε APIs διάσημων εφαρμογών ηλεκτρονικού εμπορίου. Στο τρίτο κεφάλαιο αναλύουμε τις μεθοδολογίες αποθήκευσης δεδομένων που ακολουθεί μια διαδικτυακή εφαρμογή ηλεκτρονικού εμπορίου. Στο τέταρτο κεφάλαιο αναλύουμε τη δομή της βάσης δεδομένων που χρησιμοποιήσαμε, τους πίνακες και τα πεδία που αυτοί έχουν. Στο πέμπτο κεφάλαιο περιγράφουμε τις βασικές λειτουργίες της εφαρμογής. Στο έκτο κεφάλαιο παραθέτουμε τον κώδικα που υλοποιεί τις συναρτήσεις επιλογής καταστημάτων Στο έβδομο κεφάλαιο περιγράφουμε τη διαδικασία που ακολουθήσαμε για να ανεβάσουμε την εφαρμογή που κατασκευάσαμε σε μια υπηρεσία υπολογιστικού νέφους (cloud computing), το heroku.

Μέρος Ι

APIs εφαρμογών ηλεκτρονικού εμπορίου

Κεφάλαιο 2

Δημοφιλή APIs εφαρμογών ηλεκτρονικού εμπορίου

2.0.1 Skroutz

Το Skroutz είναι μια μηχανή αναζήτησης για σύγκριση προϊόντων μεταξύ δεκάδων ηλεκτρονικών καταστημάτων στην Ελλάδα. Ξεκίνησε το 2004 με μόλις 15 ηλεκτρονικά καταστήματα και 40.000 προϊόντα. Τώρα το skroutz είναι η μεγαλύτερη μηχανή αναζήτησης προϊόντων στην Ελλάδα με 2 εκατομύρια μοναδικούς χρήστες και 53 εκατομύρια επισκέψεις το μήνα. Απασχολεί πάνω από 40 άτομα και έχει επεκταθεί και στην Τουρκία με το όνομα alve.com. Επίσης έχει ξεκινήσει να προσφέρει και μια πλατφόρμα κατασκευής e-shop, το skroutzstore.gr. Είναι φτιαγμένο σε γλώσσα Ruby με το Ruby on Rails, ένα framework κατασκευής διαδικτυακών εφαρμογών ανοιχτού κώδικα που είναι ιδανικό για την γρήγογη ανάπτυξη εφαρμογών.

Το API του προσφέρει πρόσβαση σε πολλά απ τα δεδομένα του skroutz. Για να το χρησιμοποιήσει ο προγραμματιστής πρέπει να αποκτήσει id χρήστη και κωδικό από το skroutz. [14] Η πρόσβαση γίνεται μέσω HTTPS, από το api.skroutz.gr και όλα τα δεδομένα αποστέλονται και λαμβάνονται σαν JSON.

Ενέργειες API

- Category
  Η κατηγοριοποίηση του Skroutz.gr αποτελείται από ένα δέντρο περίπου 800 κατηγορίων. Τα προιόντα και τα Skus ανήκουν υποχρεωτικά σε κάποια τελική κατηγορία, μια κατηγορία δηλαδή χωρίς υποκατηγορίες.

Υπάρχουν δύο τύποι κατηγοριών, αυτές που περιέχουν Sku και αυτές που περιέχουν αταξινόμητα προιόντα.

  - About
  - List all categories
  - Retrieve a single category
  - Retrieve the parent of a category
  - Retrieve the root category
  - List the children categories of a category
  - List a category’s manufacturers

- SKU
  Το Sku είναι ένα γενικό προϊόν που ομαδοποίει όμοια προιόντα ανάμεσα στα καταστήματα. Το Sku μπορεί να περιέχει κριτικές ή specifications, όπως π.χ. τον επεξεργαστή στην περίπτωση ενός laptop ή τις ίντσες στην περίπτωση μιας τηλεόρασης.

  - About
  - List SKUs of specific category
  - Ordering
  - Filtering
  - Retrieve a single SKU
  - Retrieve similar SKUs
  - Retrieve a SKU’s products
  - Retrieve a SKU’s reviews
  - Retrieve a SKU’s specifications

- Product
  Το Product αναφέρεται σε ένα συγκεκριμένο προϊόν κάποιου καταστήματος.
  Η ομάδα του Skroutz.gr στις περισσότερες δημοφιλείς κατηγορίες ομαδοποιεί ίδια προιόντα που υπάρχουν σε διαφορετικά καταστήματα. Η ομαδοποίηση αυτή ονομάζεται Sku.

  - Retrieve a single product

- Shop

  - Retrieve a single shop
  - Retrieve a shop’s reviews
  - List shop locations
  - Retrieve a single shop location
  - Search for shops

- Manufacturer
  - List manufacturers
  - Retrieve a single manufacturerSearch
  - Retrieve a manufacturer’s categories
  - Retrieve a manufacturer’s SKUs

2.0.2 Ebay

Το eBay είναι μια ιστοσελίδα όπου λαμβάνουν χώρα διαδικτυακές δημοπρασίες και πωλήσεις. Ο καθένας μπορεί να αγοράσει κάποιο προϊόν ή να δημιουργήσει ένα διαδικτυακό κατάστημα στο ebay και να πουλάει τα προϊόντα του σε όλο το κόσμο. Μεγάλο μέρος των αγορών γίνεται μέσω του Paypal, το οποίο εξαγοράστηκε από το ebay, και όχι άμεσα με κάρτες τραπεζών.

Το API του ebay επιτρέπει στους προγραμματιστές να αλληλεπιδράσουν άμεσα με τη βάση προϊόντων του. Με το API του ebay μπορούν οι προγραμματιστές να κάνουν τα εξής:

- Εμφάνιση προϊόντων
- Εμφάνιση της μεγαλύτερης προσφοράς για μια δημοπρασία
- Εμφάνιση κατηγοριών προϊόντων
- Δημιουργία σχολίων μετά την ολοκλήρωση μιας αγοράς
- Εμφάνιση προϊόντων που πουλάει κάποιος χρήστης
- Εμφάνιση δημοπρασίών στις οποίες έχει συμμετάσχει κάποιος
- Καταχώριση προϊόντος στο eBay
- Εμφάνιση πληροφοριών για κάποιο προϊόν

Το ebay έχει 4 διαφορετικά API

- Trading Api
- Finding Api
- Shopping Api
- Feedback Api
- Open Api

Το API που αφορά την εργασία μας είναι το Finding API και το Shopping API

Ενέργειες API

- Finding API
  The Finding API enables developers to access eBay's next generation search capabilities. The Finding API enables developers to search for items using eBay's next generation Finding Platform. Buying applications can build search and browse experiences using the powerful search and search refinement capabilities offered by the Finding APIs.
  - getSearchKeywordsRecommendation: Get recommended keywords for search
  - findItemsByKeywords: Search items by keywords
  - findItemsByCategory: Search items in a category
  - findItemsAdvanced: Advanced search capabilities
  - findItemsByProduct: Search items by a product identifier
  - findItemsIneBayStores: Search items in stores
  - getHistograms: Get category and domain meta data
- Shopping API
  eBay Shopping APIs support performance-optimized, lightweight APIs for accessing public eBay data in the Web 2.0 world. The Shopping API is optimized for response size, speed and usability. Search for eBay items, products and reviews, user info, and popular items and searches. Retrieve public eBay data in a buyer-friendly view, for easy consumption by widgets, search tools, and other buyer-focused applications.
  - FindProducts: Search for products on eBay via keyword or ProductID
  - FindHalfProducts: Search for Half.com product-related information from the catalog via keywords and category based search

- GetSingleItem: Simplified buyer specific view of item data
- GetItemStatus: Retrieve changing item information towards the end of an auction
- GetShippingCosts: Retrieve item shipping details
- GetMultipleItems: Retrieve multiple item data in a single request
- GetUserProfile: Retrieve eBay user profile and feedback information
- FindPopularSearches: Retrieve popular and related search keywords by category or keyword
- FindPopularItems: Retrieve popular items by category
- FindReviewsandGuides: Search for reviews and guides by user, category or ProductID
- GetCategoryInfo: Retrieve category information including child categories and breadcrumbs
- GeteBayTime: Retrieve the current eBay Official Time

2.0.3 Amazon

Η Amazon.com είναι η μεγαλύτερη εταιρεία στον κόσμο από την άποψη του κύκλου εργασιών από την πώληση αγαθών και υπηρεσιών μέσω του Διαδικτύου και μια από τις πρώτες που βασίστηκε στο Διαδίκτυο γα την παροχή των υπηρεσιών. [20]

Το API που έχει η Amazon το οποίο αλληλεπιδρά με τη βάση δεδομένων των προϊόντων της το ονομάζει Amazon Product Advertising API. Μέσω του API οι προγραμματιστές έχουν τη δυνατότητα να κάνουν αναζητήσεις προϊόντων στον κατάλογο του Amazon, να δούν πληροφορίες για κάποιο συγκεκριμένο προϊόν και να αλληλεπιδράσουν με το καλάθι προϊόντων κάποιου χρήστη. [19]

Μάλιστα για να παροτρύνει η Amazon προγραμματιστές να χρησιμοποιούν το API της για να φτιάχνουν εφαρμογές, δίνει στον κάτοχο της εφαρμογής μερίδιο 8,5% από τα κέρδη όταν κάποιος χρήστης αγοράσει προϊόν μέσω αυτής. [1]

Ενέργειες API

- ListMatchingProducts: Returns a list of products and their attributes, based on a search query.

- GetMatchingProduct: Returns a list of products and their attributes, based on a list of ASIN values.
- GetMatchingProductForId: Returns a list of products and their attributes, based on a list of ASIN, GCID, SellerSKU, UPC, EAN, ISBN, and JAN values.
- GetCompetitivePricingForSKU: Returns the current competitive price of a product, based on SellerSKU.
- GetCompetitivePricingForASIN: Returns the current competitive price of a product, based on ASIN.
- GetLowestOfferListingsForSKU: Returns pricing information for the lowest-price active offer listings for a product, based on SellerSKU.
- GetLowestOfferListingsForASIN: Returns pricing information for the lowest-price active offer listings for a product, based on ASIN.
- GetMyPriceForSKU: Returns pricing information for your own offer listings, based on SellerSKU.
- GetMyPriceForASIN: Returns pricing information for your own offer listings, based on ASIN.
- GetProductCategoriesForSKU: Returns the parent product categories that a product belongs to, based on SellerSKU.
- GetProductCategoriesForASIN: Returns the parent product categories that a product belongs to, based on ASIN.
- GetServiceStatus: Returns the operational status of the Products API section*.

2.1 Σύγκριση APIs

API	Τεκμηρίωση API	API Endpoint	Πρωτόκολα	Αυθεντικοποίηση
http://skroutz.gr	http://docs.skroutz.gr/apiv3/	https://api.skroutz.gr	JSON	OAuth2.0
http://ebay.com	https://go.developer.ebay.com/developers/ebay	http://developer.ebay.com/webservices/latest/eBaySvc.wsdl	XML, JSON, SOAP, HTTP, REST	eBay AppID
http://amazon.com	http://docs.developer.amazonservices.com/en_US/products/index.html	http://webservices.amazon.com/	XML, REST, SOAP	Shared secret

Πίνακας 2.1: Σύγκριση APIs εφαρμογών ηλεκτρονικού εμπορίου

2.2 Άλλα APIs

Shopping

Με το API του shopping.com ο προγραμματιστής έχει τη δυνατότατα να αλληλεπιδράσει με τη μεγάλη βάση προϊόντων του site. Το Shopping.com είναι μια εταιρία του ebay και μια από τις μεγάλες εταιρίες στη κατηγορία μηχανών αναζήτησης και σύγκρισης προϊόντων. Το API δίνει πρόσβαση σε προσφορές πάνω από 6000 πωλητών, αναζητήσεις προϊόντων, χαρακτηριστηκά και πληροφορίες προϊόντων και τιμές για προϊόντα από διαφορετικούς πωλητές. [9]

Shopzilla

Το Shopzilla API δίνει πρόσβαση στον μεγάλο κατάλογο προϊόντων του site, σε πάνω από 8000 πωλητές, βαθμολογίες για πωλητές και προϊόντα και τιμές. Επίσης η εταιρεία δίνει τη δυνατότητα στους προγραμματιστές που θα χρησιμοποιήσουν το API της για να φτιάξουν εφαρμογές, να βγάλουν κέρδος από αυτές. Επίσης το Shopzilla διαθέτει και ξεχωριστά APIs για κάποιες χώρες, συγκεκριμένα την Αγγλία τη Γαλία και τη Γερμανία. [10]

Etsy

Το Etsy είναι μια διαδικτυακή πλατφόρμα για αγοραπωλησίες χειροποίητων προϊόντων. Το API του επιτρέπει την ανάπτυξη εφαρμογών για το etsy. [6]

Groupon

Το Groupon είναι ένα site με ημερήσιες προσφορές για προϊόντα και υπηρεσίες σε πολλές πόλεις των Ηνωμένων Πολιτειών. Τα τελευταία χρόνια έχει επεκταθεί και σε πολλές πόλεις της Ευρώπης. Το API του Groupon επιτρέπει την άμεση πρόσβαση στις προσφορές που διαθέτει. [7]

Shopify

Το Shopify είναι μια πλατφόρμα κατασκευής διαδικτυακών καταστημάτων. Απευθύνεται σε πωλητές που είτε θέλουν να ανοίξουν ένα διαδικτυακό κατάστημα, είτε έχουν ένα φυσικό κατάστημα και θέλουν να αποκτήσουν ένα διαδικτυακό πρόσωπο και να κάνουν πωλήσεις μέσω διαδικτύου. Όλες οι ενέργειες που μπορέι να κάνει κάποιος χρήστης στο site είναι δυνατό να

πραγματοποιηθούν και μέσω του API που διαθέτει.Τα δεδομένα που μπορεί να αλληλεπιδράσει κάποιος μέσω του API, αφορούν προϊόντα, πελάτες, ειδοποιήσεις και άλλα. [8]

Μέρος ΙΙ

Κατασκευή Εφαρμογής Ηλεκτρονικού Εμπορίου

Κεφάλαιο 3

Μέθοδολογία αποθήκευσης δεδομένων

Μια διαδικτυακή εφαρμογή ηλεκτρονικού εμπορίου χρειάζεται να διαχειρίζεται και να αποθηκεύει δεδομένα. Τα δεδομένα αυτά μπορεί να περιέχουν προϊόντα, καταστήματα, SKUs (stock keeping units) και ότι άλλο χρειάζεται η εφαρμογή. Υπάρχουν οι εξής μέθοδοι που μπορούν αυτά τα δεδομένα να αποθηκευτούν

3.1 Απλά αρχεία κειμένου

Ως απλά αρχεία κειμένου εννοούμε αρχεία τα οποία μπορούμε να τα ανοίξουμε με έναν απλό κειμενογράφο και περιέχουν κείμενο σε μορφή που διαβάζεται από τον άνθρωπο. Πράγμα που δεν ισχύει για τα δυαδικά (binary) αρχεία δεδομένων. Παρόλο που αυτή η μέθοδος αποθήκευσης είναι απλή και το περιεχόμενο των αρχείων κατανοητό στον άνθρωπο, δεν είναι πολύ χρήσιμη για τον υπολογιστή ο οποίος χρειάζεται μια πιο δομημένη μορφή δεδομένων για να τα επεξεργαστεί. Παράδειγμα πιο δομημένης μορφής δεδομένων απλού κειμένου θα μπορούσε να ήταν σε ένα αρχείο με προϊόντα το κάθε προϊόν να βρίσκεται σε διαφορετική γραμμή όπως στα αρχεία CSV που θα δούμε στην επόμενη ενότητα.

3.2 Δομημένα αρχεία κειμένου

Τα δομημένα αρχεία κειμένου είναι απλά αρχεία κειμένου που τα δεδομένα που περιέχουν ακολουθούν ένα κοινό πρότυπο το οποίο ο υπολογιστής μπορεί να αναγνωρίσει και να επεξεργαστεί. Συχνά χρησιμοποιούμενες μορφές δομημένων αρχείων κειμένου είναι τα αρχεία CSV, XML και JSON.

3.2.1 Comma separated values (CSV)

Τα αρχεία αποθηκεύουν δεδομένα του ίδιου τύπου, π.χ. μόνο προϊόντα ή μόνο καταστήματα. Κάθε αντικείμενο πρέπει να είναι σε διαφορετική γραμμή και οι ιδιότητές του να χωρίζονται με έναν ειδικό χαρακτήρα. π.χ. κόμμα ή ερωτηματικό. [16]

3.2.2 Extensible Markup Language (XML)

Τα αρχεία xml χρησιμοποιούνται ευρέως στο διαδίκτυο. Η Extensible Markup Language ορίζει τους κανόνες με τους οποίους συντάσσονται αυτά τα αρχεία. Η σύνταξη περιέχει Tags, Elements και Attributes.

Tag Το tag ξεκινάει με το χαρακτήρα < και τελειώνει με >. Τα tags συνήθως χρησιμοποιούνται σε ζεύγη και περικλείουν ένα element όπου τότε το 2ο tag ξεκινάει με </ Για παράδειγμα: `<product></product>`

Element Το element είναι η πληροφορία που βρίσκεται ανάμεσα σε ένα tag ανοίγματος και σε ένα tag κλεισίματος. Για παράδειγμα: `<product>iphone</product>`

Attribute Το attribute ειναι μια ιδιότητα που έχει ένα tag. συντάσεται ως ιδιότητα=τιμή μέσα σε ένα tag. Για παράδειγμα: `<img src="image.jpg"/>` όπου το tag είναι το img, η ιδιότητα το src και η τιμή της ιδιότητας το "image.jpg"

Στην περίπτωση μιας διαδικτυακής εφαρμογής ηλεκτρονικού εμπορίου ένα αρχείο xml που περιέχει προϊόντα θα μπορούσε να έχει την εξής μορφή: [17]

```
<?xml version="1.0" encoding="UTF-8"?>
<products>
  <product>
    <name>iphone 3</name>
    <price>300</price>
    <store>e-shop 1</store>

    <name>iphone 3</name>
    <price>320</price>
    <store>e-shop 2</store>

    <name>ipad</name>
```

```
    <price>400</price>
    <store>e-shop 1</store>

    <name>ipad</name>
    <price>430</price>
    <store>e-shop 3</store>
  </product>
</products>
```

3.2.3 JavaScript Object Notation (JSON)

Τα αρχεία json όπως και τα xml χρησιμοποιούνται ευρέως στο διαδίκτυο και συγκεκριμένα για την ανταλλαγή δεδομένων μεταξύ ενός server και ενός client μέσω ενός API. Ένα πλεονέκτημα που έχουν τα δεδομένα σε μορφή JSON είναι ότι μπορούν να προσπελαστούν και να επεξεργαστούν άμεσα από κώδικα JavaScript. Για παράδειγμα ο κώδικας:

```
var p = JSON.parse(product);
```

διαβάζει το κείμενο σε μορφή JSON που βρίσκεται στη μεταβλητή product, το μετατρέπει σε αντικείμενο της JavaScript και το αναθέτει στη μεταβλητή p. [23] Στην περίπτωση μιας διαδικτυακής εφαρμογής ηλεκτρονικού εμπορίου ένα αρχείο JSON που περιέχει προϊόντα θα μπορούσε να έχει την εξής μορφή:

```
{
  "products": [
    {
      "name": "iphone 1",
      "price": 100
    },
    {
      "name": "iphone 2",
      "price": 200
    },
    {
      "name": "iphone 3",
      "price": 300
    }
  ]
}
```

3.3 Σχεσιακές βάσεις δεδομένων

Οι βάσεις δεδομένων είναι συστήματα οργανωμένης αποθήκευσης δεδομένων που αφορούν μια συλλογή αντικείμενων με σκοπό να μπορούν να εκτελεστούν ενέργειες όπως εισαγωγή ενός αντικειμένου στη συλλογή, επεξεργασία και αναζήτηση δεδομένων με βάση κάποια κριτήρια. Σε σχέση με τους προηγούμενους τύπους αποθήκευσης που αναφέραμε προηγουμένως οι βάσεις δεδομένων αποθηκεύουν τα δεδομένα τους σε δυαδικά (binary) αρχεία και όχι σε απλά αρχεία κειμένου με αποτέλεσμα να μην μπορούν να διαβαστούν και να επεξεργαστούν απ ευθείας από τον απλό χρήστη αλλά να χρειάζεται ειδικό λογισμικό. Το λογισμικό αυτό παρέχει τις απαραίτητες διεπαφές χρήσης που μπορεί να είναι ένα πρόγραμμα που εκτελεί εντολές σε κονσόλα, μια γραφική διεπαφή, ή προγραμματιστικές βιβλιοθήκες που επιτρέπουν την εκτέλεση ενεργειών στη βάση μέσα από κώδικα. [22]

Η πλειοψηφία των σχεσιακών βάσεων δεδομένων είναι συμβατές με τη γλώσσα SQL (Structured Query Language). Αυτό σημαίνει ότι ο προγραμματιστής μπορεί να αλλάξει τη βάση δεδομένων που χρησιμοποιεί πχ. από Oracle να χρησιμοποιήσει MySQL και το πρόγραμμα του να συνεχίζει να λειτουργεί χωρίς αλλαγες αφού επικοινωνεί με τη βάση μέσω διεπαφής SQL.

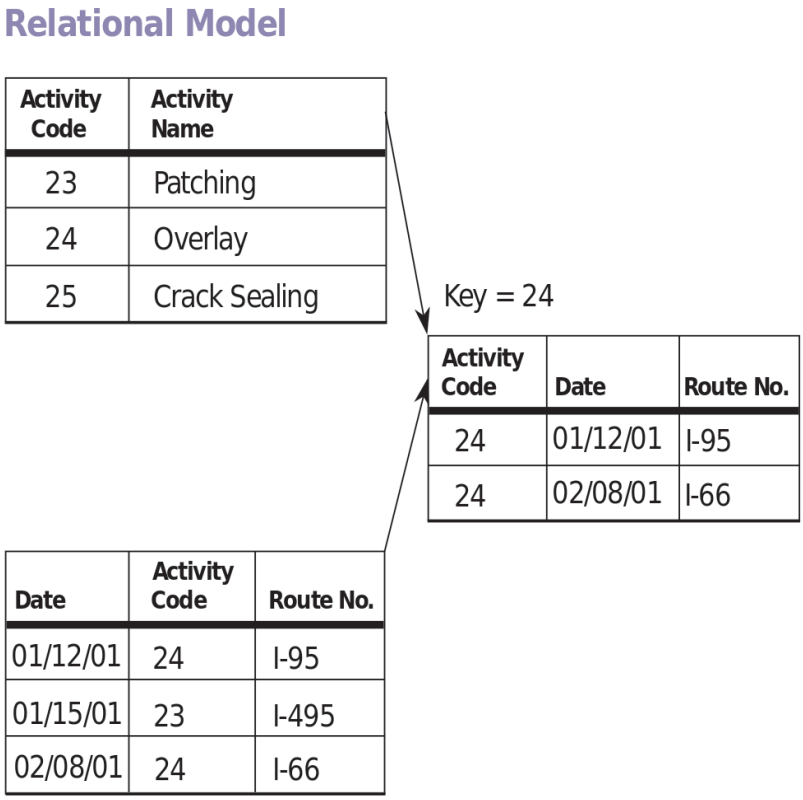
Στο σχεσιακό μοντέλο δεδομένων τις συλλογές αντικειμένων που θέλουμε να αποθηκεύσουμε στη βάση τα χωρίζουμε σε πίνακες ανάλογα με το τι αντιπροσωπεύουν και κάθε αντικείμενο καταλαμβάνει μια γραμμή στον πίνακα. Για παράδειγμα σε μια βάση που θέλουμε να αποθηκεύσουμε προϊόντα και καταστήματα θα τα χωρίσουμε σε δύο πίνακες, έναν για τα προϊόντα και έναν για τα καταστήματα και σε κάθε πίνακα κάθε προϊόν ή κατάστημα θα καταλαμβάνει μια γραμμή. Όπως στο σχήμα 4.1

3.4 Άλλοι τύποι βάσεων δεδομένων

Εκτός από τις σχεσιακές βάσεις δεδομένων υπάρχουν και άλλοι τύποι βάσεων δεδομένων που δημιουργήθηκαν για να εξυπηρετήσουν ειδικές περιπτώσεις ή να καλύψουν ανάγκες στις οποίες οι σχεσιακές βάσεις δεδομένων υστερούν. Ενδεικτικά αναφέρουμε τις κυριότερες κατηγορίες μη σχεσιακών βάσεων

Column Αυτή η κατηγορία βάσεων δεδομένων καλύπτει την ανάγκη για κατανεμημένες βάσεις δεδομένων με μεγάλο όγκο δεδομένων. Παραδείγματα τέτοιων συστημάτων: Accumulo, Cassandra, Druid, HBase

Document Οι Αποθήκες Δεδομένων Εγράφων χρησιμοποιούν αντί για πίνακες και εγγραφές τις συλλογές εγγράφων και τα έγγραφα αντίστοιχα. Παραδείγματα τέτοιων συστημάτων:



Σχήμα 3.1: Παράδειγμα τυπικής σχεσιακής βάσης δεδομένων. Κάθε πίνακας περιέχει διαφορετικές πληροφορίες για τα αντικείμενα σε στήλες. Κάθε γραμμή περιέχει ένα αντικείμενο. Διαφορετικά αντικείμενα συσχετίζονται μεταξύ τους με βάση το πεδίο κλειδί που έχει κάθε αντικείμενο. [4]

Clusterpoint, Apache CouchDB, Couchbase, MarkLogic, MongoDB

Key-value Οι Αποθήκες Δεδομένων Κλειδιού-Τιμής αποθηκεύουν τα δεδομένα με ένα πεδίο-κλειδί το οποίο είναι μοναδικό. Αυτή η απλή προσέγγιση προσφέρει υψηλές ταχήτητες αποθήκευσης και ανάκτησης δεδομένων με βάση το κλειδί τους. Παραδείγματα τέτοιων συστημάτων: Dynamo, Redis, FoundationDB, MemcacheDB, Riak, FairCom c-treeACE

Graph Οι Αποθήκες Δεδομένων Γράφων αποθηκεύουν τα δεδομένα ως κόμβους γράφου και τις σχέσεις μεταξύ των δεδομένων ως συνδέσεις του γράφου. Παραδείγματα τέτοιων συστημάτων: Allegro, Neo4J, InfiniteGraph, OrientDB, Virtuoso, Stardog

3.5 Object relational mapping

Στις αντικειμενοστρεφής γλώσσες προγραμματισμού όπως η C++ η Java η Ruby και πολλές άλλες, ο χειρισμός των σχεζόμενων δεδομένων και των διαδικασιών που επενεργούν σε αυτά γίνεται μέσω μίας δομής δεδομένων που τα περιβάλλει ως αυτόνομη οντότητα με ταυτότητα και δικά της χαρακτηριστικά. Αυτή η δομή δεδομένων καλείται αντικείμενο ή κλάση. [24]

Τα δεδομένα μιας διαδικτυακής εφαρμογής ηλεκτρονικού εμπορίου λοιπόν μπορούν να μοντελοποιηθούν με αντικείμενα αντικειμενοστρεφούς προγραμματισμού. Για παράδειγμα θα μπορούσαμε να έχουμε `Product` και `Store`. Η κλάση `Store` θα μπορούσε να έχει μια μεταβλητή `name` τύπου κειμένου και μια μεταβλητή `products` τύπου array με ένα array από πολλά αντικείμενα `Product`

Για την αποθήκευση ή ανάκτηση αυτών των αντικείμενων σε μια βάση δεδομένων μπορούμε να χρησιμοποιήσουμε απ' ευθείας εντολές SQL μέσα στο πρόγραμμά μας.

Για παράδειγμα η παρακάτω κλήση της SQL εντολής Select μπορεί να μας επιστρέψει όλα τα προϊόντα από τον πίνακα `products`
`SELECT * FROM products;` Το μειονέκτημα της χρήσης SQL μέσα στο πρόγραμμά μας είναι ότι η SQL δε θα μας επιστρέψει τα δεδομένα σαν αντικείμενα ωστε να τα επεξεργαστούμε απ' ευθείας.

Λύση σε αυτό το πρόβλημα δίνουν οι βιβλιοθήκες ORM (Object Relational Mapping). Αυτές οι βιβλιοθήκες μας επιτρέπουν να αποθηκεύουμε και να ανακτούμε αντικείμενα από τη βάση δεδομένων μέσω εντολών που προσφέρουν και όχι επερωτήματα SQL. [11]
Για παράδειγμα ο παρακάτω κώδικας java: [15]

```
String sql = "SELECT ... FROM products WHERE id = 10";
```

```
DbCommand cmd = new DbCommand(connection, sql);
Result res = cmd.Execute();
String name = res[0]["NAME"];
```

Μπορεί να γίνει: [15]

```
Product p = repository.GetProduct(10);
String name = p.Name;
```

ή σε μερικά συστήματα ORM μπορεί να γίνει πιο απλά: [15]

```
Product p = Product.Get(10);
```

Παραδείγματα συστημάτων ORM είναι το LiteSQL για C++, το Hibernate για Java, το Zend για PHP και το ActiveRecord για Ruby το οποίο θα δούμε στη συνέχεια.

3.5.1 ActiveRecord Ruby ORM

Η ένοια του Migration

Το ActiveRecord έχει την ένοια του Migration. Κάνοντας ένα Migration ο χρήστης πραγματοποιεί αλλαγές στη βάση δεδομένων που χρησιμοποιείται. Τέτοιες αλλαγές μπορεί να είναι για παράδειγμα η πρόσθεση μιας στήλης σε ένα πίνακα, ή η δημιουργία ενός καινούργιου πίνακα κ.α. Για παράδειγμα ο κώδικας

```ruby
class CreateProductTable < ActiveRecord::Migration
  def change
    create_table :products do |t|
      t.string  :name   # πεδίο name τύπου κειμένου
      t.decimal :price  # πεδίο price τύπου δεκαδικού
    end
  end
end
CreateProductTable.new.migrate(:change)
```

δημιουργεί μια καινούρια κλάση `CreateProductTable` τύπου `ActiveRecord::Migration` η οποία περιγράφει τις στήλες που θα έχει ο πίνακας `products` που θέλουμε να δημιουργηθεί. Μετά η εντολή

```
CreateProductTable.new.migrate(:change)
```

που ακολουθεί δημιουργεί τον πίνακα στη βάση. [12]

Αντικειμένο ActiveRecord

Δημιουργία Για να δημιουργήσουμε ένα αντικείμενο ActiveRecord απλά δηλώνουμε ότι η κλάση του αντικειμένου είναι τύπου `ActiveRecord::Base`. Για παράδειγμα για να ορίσουμε και να δημιουργήσουμε ένα κενό αντικείμενο `Product` χρησιμοποιούμε τον κωδικα: [11]

```
class Product < ActiveRecord::Base

end
p = Product.new()
```

Επεξεργασία Το αντικείμενο Product που ορίσαμε προηγουμένως όντας ένα αντικείμενο ActiveRecord αποκτά κάποιες συναρτήσεις που επιδρούν πάνω σε αυτό και κάποιες μεταβλητές. Συγκεκριμένα αποκτά τις μεταβλητές name και price. Αυτές τις μεταβλητές δεν τις ορίσαμε εμείς στην κλάση Product αλλά το ActiveRecord τις ορίζει επειδή βλέπει ότι στη βάση δεδομένων στον πίνακα Products υπάρχουν οι στήλες name και price.

Έτσι σε συνέχεια του προηγούμενου κώδικα για να επεξεργαστούμε το προϊόν που βρίσκεται στη μεταβλητή p μπορούμε να κάνουμε:

```
p.name = "iphone"
p.price = 100.45
```

τώρα το προϊόν που βρίσκεται στη μεταβλητή p έχει λάβει τιμές στις μεταβλητές name και price αλλά αυτές οι τιμές δεν έχουν αποθηκευτεί στη βάση δεδομένων. [11]

Αποθήκευση Εκτος από τις μεταβλητές που ορίζει αυτόματα το ActiveRecord σε ένα προϊόν ορίζει αυτόματα και κάποιες συναρτήσεις. Μια από αυτές συγκεκριμένα είναι η συνάρτηση

save() η οποία αποθηκεύει τυχόν αλλαγές σε ένα αντικείμενο ή ένα αντικείμενο που μόλις έχει δημιουργηθεί στη βάση δεδομένων. [11]

```
p.save()
```

Διαγραφή Άλλη μια συνάρτηση που το ActiveRecord ορίζει αυτόματα στα αντικείμενα είναι η συνάρτηση destroy() η οποία διαγράφει το αντικείμενο που την κάλεσε από τη βάση δεδομένων. Σε συνέχεια του προηγούμενου κώδικα, για να διαγράψουμε το προϊόν που βρίσκεται στη μεταβλητή p χρησιμοποιούμε τον κώδικα: [11]

```
p.destroy()
```

Αναζήτηση Εκτος από τις συναρτήσεις και μεταβλητές που ορίζει αυτόματα το ActiveRecord στα προϊόντα ή ίδια η κλάση Product αποκτά κάποιες συναρτήσεις οι οποίες αφορούν το σύνολο των προϊόντων που βρίσκονται στον πίνακα products της βάσης δεδομένων. Μια από αυτές, τη συνάρτηση Product.new() την είδαμε όταν δημιουργήσαμε ένα αντικείμενο product.

Για να αναζητήσουμε προϊόντα από τη βάση δεδομένων το Activerecord διαθέτει παραπάνω από μία συναρτήσεις: [13]

Product.find(key) Παίρνει σαν όρισμα ένα ακέραιο αριθμό που αντιστοιχεί στο id του προϊόντος και επιστρέφει το προϊόν

Product.first() Επιστρέφει το πρώτο προϊόν του πίνακα

Product.last() Επιστρέφει το τελευταίο προϊόν του πίνακα

Product.find_by(contition) Δεχεται σαν όρισμα ένα Hash με μια συνθήκη, πχ name: 'iphone' και επιστρέφει ένα προϊόν

Product.where(condition, arg) Δεχεται σαν όρισμα ένα Hash με μια συνθήκη, πχ name: 'iphone' και επιστρέφει πολλά προϊόντα

Product.where(condition, args...) Δεχεται σαν όρισμα ένα String με μια συνθήκη επερωτήματος SQL και μια παράμετρο για τη συνθήκη και επιστρέφει ένα Array με πολλά προϊόντα.

Product.all() Επιστρέφει όλα τα προϊόντα.

Παραδείγματα χρήσης των παραπάνω εντολών:

```ruby
Product.find(1) # επιστρέφει το προϊόν με id 1
Product.first() # επιστρέφει το πρώτο προϊόν
Product.last()  # επιστρέφει το τελευταίο προϊόν

# επιστρέφει ένα προϊόν με name iphone
Product.find({:name => "iphone"})

# επιστρέφει όλα τα προϊόντα με name iphone
Product.where({:name => "iphone"})

# επιστρέφει όλα τα προϊόντα με τιμή μεταξύ 100 και 500
Product.where("price > ? AND price < ?", 100, 500)

# Επιστρέφει όλα τα προϊόντα του πίνακα products
# και τα αναθέτει στη μεταβλητή products
products = Product.all

# Εμφάνιζει τα προϊόντα στην οθόνη κάνοντας χρήση της
# εντολής puts που εμφανίζει κείμενο στην οθόνη
# και της εντολής inspect η οποία μετατρέπει ένα
# αντικείμενο σε κείμενο
puts(products.inspect)
```

3.5.2 Ολοκληρωμένο παράδειγμα χρήσης ActiveRecord

Το παράδειγμα που ακολουθεί χρησιμοποιεί τις εντολές και το κώδικα που δείξαμε προηγουμένως για να κάνει τα εξής:

- Δημιουργία του πίνακα προϊόντων στη βάση δεδομένων
- Δημιουργία 4 προϊόντων στη βάση δεδομένων
- Εμφάνιση όλων των προϊόντων της βάσης δεδομένων

```ruby
# Βιβλιοθήκες
# Εδώ δηλώνουμε τις βιβλιοθήκες Ruby
# που θέλουμε να χρησιμοποιήσουμε
```

```ruby
require('active_record') # ActiveRecord ORM
require('sqlite3')       # διεπαφή για βάσεις δεδομένων SQLite

# Ρυθμίσεις ActiveRecord
# Εδώ υποδεικνύουμε στο σύστημα ActiveRecord τη
# βάση δεδομένων που θέλουμε να χρησιμοποιήσει
ActiveRecord::Base.establish_connection(
  :adapter  => :sqlite3,     # βάση δεδομένων τύπου SQLite
  :database => "db.sqlite3" # όνομα αρχείου βάσης δεδομένων
)

# Create Product Table Migration
# Εδώ δημιουργούμε ένα migration, δηλαδή μια κλάση τύπου
# ActiveRecord::Migration που περιέχει τις αλλαγες που
# θέλουμε να πραγματοποιηθούν στη βάση δεδομένων.
class CreateProductTable < ActiveRecord::Migration
  def change
    # η αλλαγή που θέλουμε να κάνουμε στη βάση δεδομένων
    # είναι να δημιουργήσουμε ένα καινούριο πίνακα
    # που ονομάζεται products
    create_table :products do |t|
      t.string  :name   # πεδίο name τύπου κειμένου
      t.decimal :price  # πεδίο price τύπου δεκαδικού
    end
  end
end

# Δημιουργία του πίνακα προϊόντων στη βάση δεδομένων
CreateProductTable.new.migrate(:change)

# Προϊόν
class Product < ActiveRecord::Base

end

# Δημιουργία προϊόντων στη βάση δεδομένων
Product.create( :name=>"iphone 1", :price=>100.45 )
```

```
41 Product.create( :name=>"iphone 2", :price=>200.63 )
42 Product.create( :name=>"iphone 3", :price=>300.67 )
43 Product.create( :name=>"iphone 4", :price=>400.12 )
44
45 # Εμφάνιζει τα προϊόντα στην οθόνη κάνοντας χρήση της
46 # εντολής puts που εμφανίζει κείμενο στην οθόνη
47 # και της εντολής inspect η οποία μετατρέπει ένα
48 # αντικείμενο σε κείμενο
49 puts(Product.all.inspect)
```

Κεφάλαιο 4

Δομή Βάσης Δεδομένων

Η βάση δεδομένων χρειάζεται να αποθηκεύσει τις εξής οντότητες: Χρήστες, Προϊόντα, Καταστήματα και Stock Keeping Units. Κάθε οντότητα θα αποθηκεύεται και σε διαφορετικό πίνακα. Ένας επιπλέον πίνακας, ο `user_skus` θα χρησιμοποιηθεί για να αποθηκεύσει τα SKUs που έχει προσθέσει ο κάθε χρήστης στα προϊόντα του.

4.1 Πίνακες Βάσης Δεδομένων

4.1.1 Users

Στον πίνακα Users αποθηκεύονται οι πληροφορίες των χρηστών που εγγράφονται στην εφαρμογή.

Πεδίο	Τύπος	Περιγραφή
`id`	Ακέραιος	Το πεδίο κλειδί του πίνακα που αναπαριστά τον κωδικό του χρήστη.
`name`	Κείμενο	Το όνομα του χρήστη, username.
`full_name`	Κείμενο	Το πλήρες όνομα του χρήστη, ονοματεπώνυμο.
`password_digest`	Κείμενο	Το Ruby on Rails δεν αποθηκεύει τους κωδικούς των χρηστών σε απλό κείμενο όπως τους δίνουν κατά την εγγραφή ή την είσοδό τους στην εφαρμογή [5] αλλά τους περνάει πρώτα μέσα από τη συνάρτηση hashing Bcrypt [21]

Πίνακας 4.1: Πεδία πίνακα Users

4.1.2 Skus

Στον πίνακα Skus αποθηκεύονται τα Stock Keeping Units που εισάγουμε στην εφαρμογή μέσω του API του Skroutz. Το Stock Keeping Unit αντιπροσωπεύει ένα προϊόν ανεξαρτήτως του καταστήματος στο οποίο βρίσκεται. Αντίθετα το προϊόν αντιπροσωπεύει ένα συγκεκριμένο προϊόν σε ένα συγκεκριμένο κατάστημα. Έχει συγκεκριμένη τιμή και διαθεσιμότητα. Ένα προϊόν της βάσης δεδομένων συνδέεται με ένα sku και ένα κατάστημα. Αντίθετα ένα sku συνδέεται με πολλά προϊόντα.

Πεδίο	Τύπος	Περιγραφή
`id`	Ακέραιος	Το πεδίο κλειδί του πίνακα που αναπαριστά τον κωδικό του sku.
`name`	Κείμενο	Το όνομα του sku.
`price_max`	Δεκαδικός	Η ακριβότερη τιμή του συγκεκριμένου sku.
`price_min`	Δεκαδικός	Η φθηνότερη τιμή του συγκεκριμένου sku.
`image`	Κείμενο	Ο σύνδεσμος προς το αρχείο εικόνας στο skroutz, που αντιπροσωπεύει το συγκεκριμένο sku.

Πίνακας 4.2: Πεδία πίνακα Skus

4.1.3 Shops

Στον πίνακα Shops αποθηκεύονται τα Καταστήματα που εισάγουμε στην εφαρμογή μέσω του API του Skroutz. Κάθε κατάστημα συνδέεται με ένα προϊόν.

Πεδίο	Τύπος	Περιγραφή
`id`	Ακέραιος	Το πεδίο κλειδί του πίνακα που αναπαριστά τον κωδικό του καταστήματος.
`name`	Κείμενο	Το όνομα του καταστήματος.
`link`	Κείμενο	Ο σύνδεσμος προς το διαδικτυακό ιστότοπο του καταστήματος.
`image_url`	Κείμενο	Ο σύνδεσμος προς το αρχείο εικόνας του καταστήματος στο skroutz.
`free_shipping`	Boolean	Εαν το κατάστημα προσφέρει δωρεαν μεταφορικά.
`shipping_min_price`	Δεκαδικός	Το ελάχιστο κόστος των προϊόντων για τα οποία το κατάστημα ενδέχεται να προσφέρει φωρεαν μεταφορικά.

Πίνακας 4.3: Πεδία πίνακα Shops

4.1.4 Products

Στον πίνακα Products αποθηκεύονται τα Προϊόντα που εισάγουμε στην εφαρμογή μέσω του API του Skroutz. Κάθε προϊόν συνδέεται με ένα κατάστημα και με ένα sku.

Πεδίο	Τύπος	Περιγραφή
`id`	Ακέραιος	Το πεδίο κλειδί του πίνακα που αναπαριστά τον κωδικό του προϊόντος.
`name`	Κείμενο	Το όνομα του προϊόντος. Σύμφωνα με το API του skroutz δύο προϊόντα που έχουν κοινά χαρακτηριστικά αλλά διαφέρουν σε μικρά πράγματα, π.χ. στο χρωμα, ομαδοποιούνται στο ίδιο sku. Για αυτή την περίπτωση χρειαζόμαστε το όνομα του προϊόντος αφού το ένα προϊόν μπορεί να ομομάζεται iphone red και το άλλο iphone grey
`sku_id`	Ακέραιος	Το πεδίο κλειδί του sku κάτω από το οποίο ομαδοποιείται το συγκεκριμένο προϊόν.
`shop_id`	Ακέραιος	Το πεδίο κλειδί του καταστήματος στο οποίο ανοίκει το συγκεκριμένο προϊόν.
`name`	Κείμενο	Η διαθεσιμότητα του συγκεκριμένου προϊόντος.
`name`	Δεκαδικός	Η τιμή του προϊόντος.

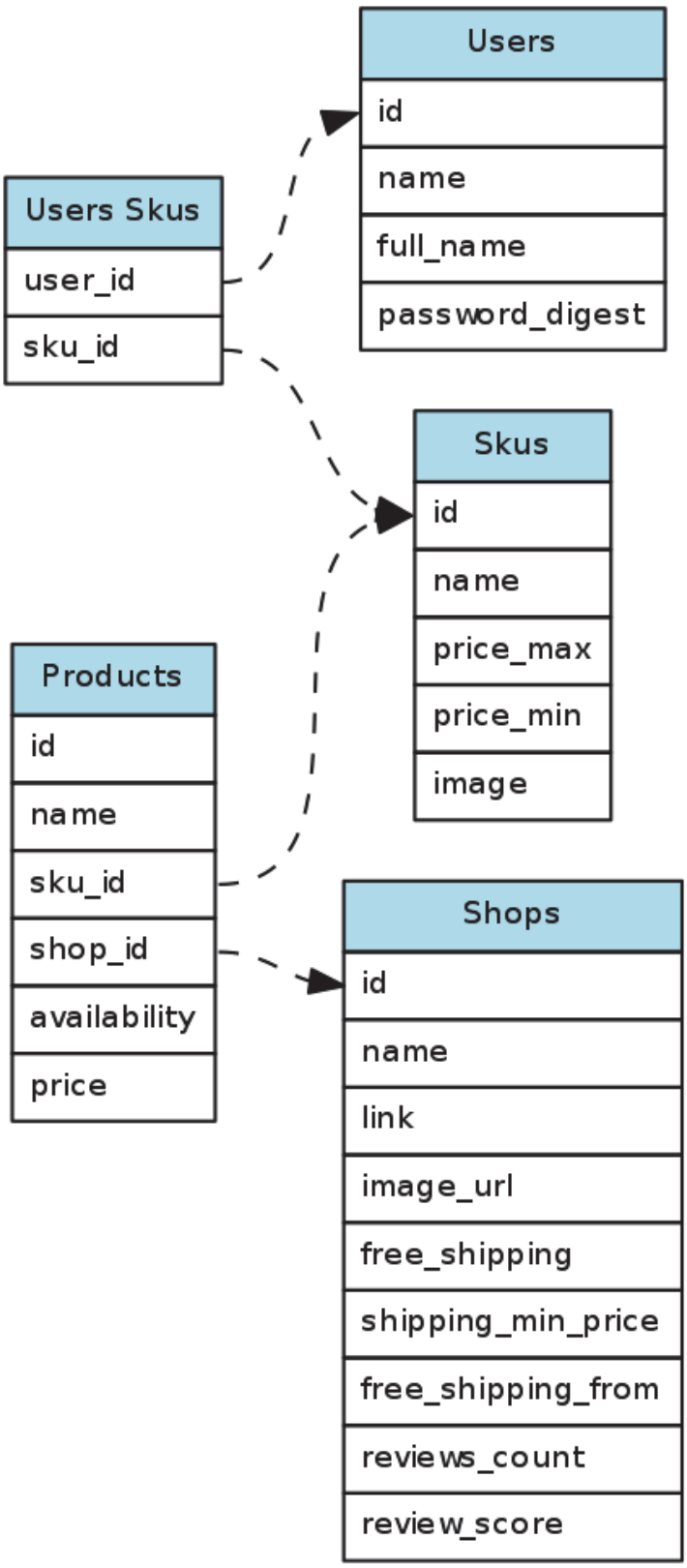
Πίνακας 4.4: Πεδία πίνακα Products

4.1.5 User_Skus

Ο πίνακας User_Skus χρειάζεται για να υποστηριχθεί η λειτουργικότητα της εφαρμογής όπου κάθε εγγεγραμένος χρήστης έχει τη δυνατότητα να προσθέσει προϊόντα στη λίστα του.

Πεδίο	Τύπος	Περιγραφή
`user_id`	Ακέραιος	Ο κωδικός του χρήστη ο οποίος έχει προσθέσει στη λίστα του το sku στο οποίο αναφέρεται το πεδίο `sku_id`.
`sku_id`	Ακέραιος	Ο κωδικός του sku.

Πίνακας 4.5: Πεδία πίνακα User_Skus



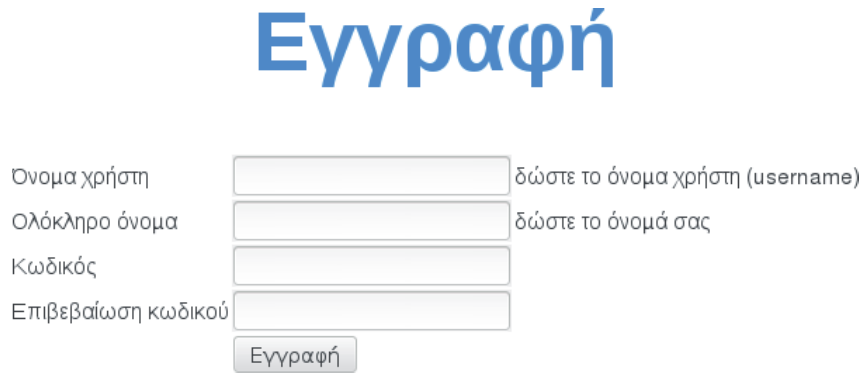
Σχήμα 4.1: Διάγραμμα βάσης δεδομένων. Η βάση δεδομένων περιέχει πέντε πίνακες. Οι σχέσεις μεταξύ των πινάκων είναι σχεδιασμένες με βέλη.

Κεφάλαιο 5

Βασικές Λειτουργίες Εφαρμογής

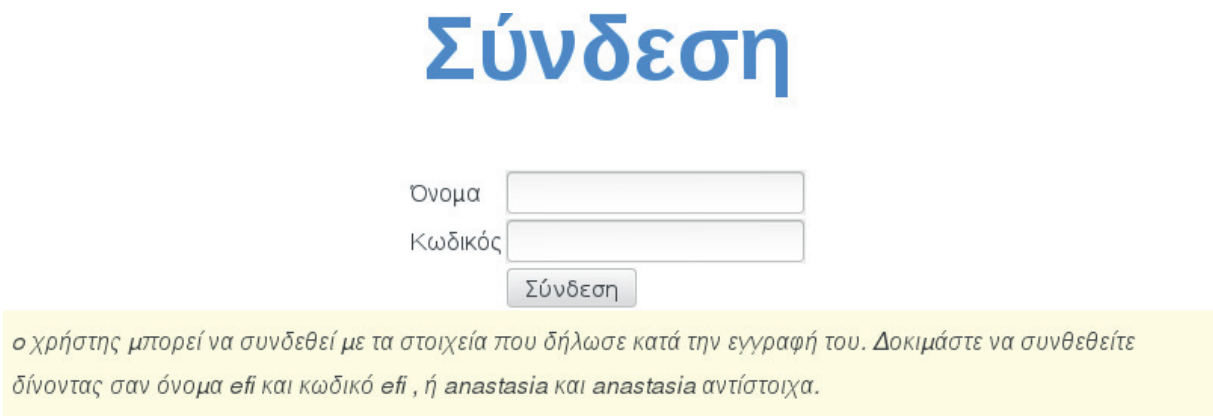
5.1 Εγγραφή και είσοδος χρήστη

Η εγγραφή κάποιου στην εφαρμογή είναι απαραίτητη ώστε να μπορέσει η εφαρμογή να κρατήσει την κατάσταση για το ποια προϊόντα έχει προσθέσει στη λίστα του. Κατά την εγγραφή γίνεται έλεγχος ώστε το όνομα (username) που δίνει ο χρήστης να μην υπάρχει ήδη στη βάση δεδομένων. Η εγγραφή γίνεται στη διεύθυνση `/signup`. Η φόρμα εγγραφής στην εφαρμογή φαίνεται στο σχήμα 5.1



Σχήμα 5.1: Φόρμα εγγραφής χρήστη στην εφαρμογή. Ο χρήστης συμπληρώνει το όνομά του (username) το ονοματεπώνυμό του και δηλώνει και ένα κλειδάριθμο.

Κατά την είσοδο του χρήστη το Ruby on Rails εγκαθιστά ένα cookie στο πρόγραμμα περιήγησης του χρήστη το οποίο περιέχει τον κωδικό που έχει ο χρήστης στη βάση δεδομένων. Η είσοδος γίνεται στη διεύθυνση `/login`. Φόρμα εισόδου χρήστη στην εφαρμογή φαίνεται στο σχήμα 5.2.



Σχήμα 5.2: Φόρμα εισόδου χρήστη στην εφαρμογή. Τα πεδία που συμπληρώνει ο χρήστης για να εισέλθει στην εφαρμογή είναι το όνομα που έχει στην εφαρμογή και ο κλειδάριθμός του.

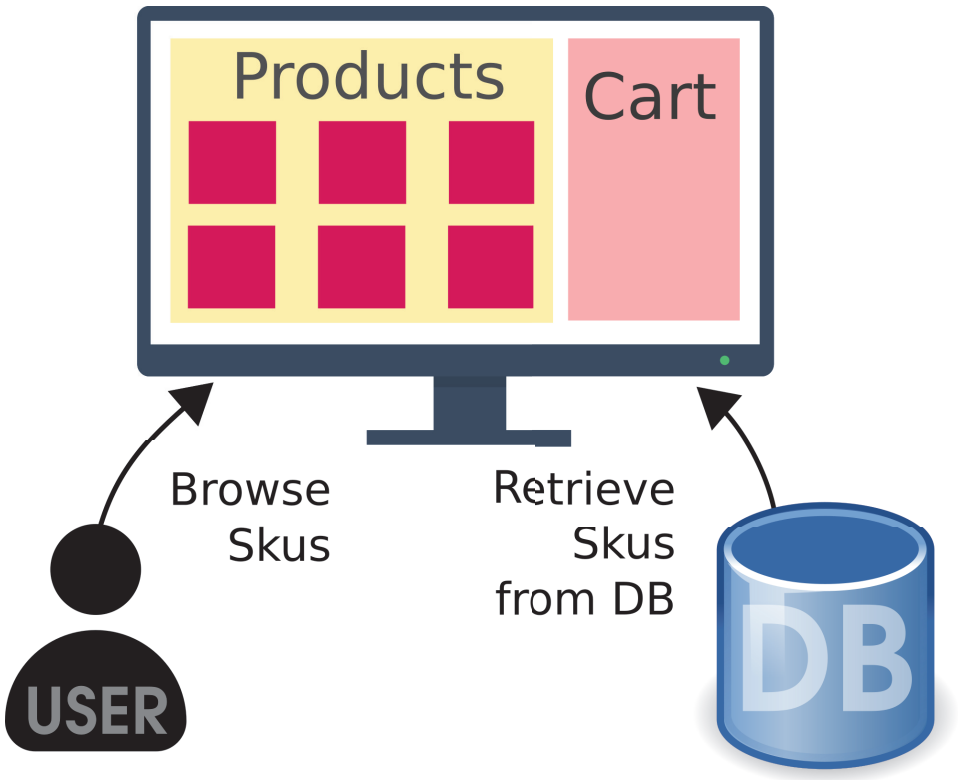
5.2 Εμφάνιση προϊόντων

Η εμφάνιση των προϊόντων ή για να ακριβολογούμε η εμφάνιση των SKUs γίνεται από την κεντρική σελίδα της εφαρμογής. Εδώ, όπως φαίνεται και στην εικόνα 5.3, για κάθε sku εμφανίζεται το όνομά του, η ανώτερη και η κατώτερη τιμή με την οποία μπορούμε να το βρούμε στα καταστήματα και αν ο χρήστης έχει κάνει είσοδο στην εφαρμογή, φαίνεται και ένα κουμπί με το οποίο μπορεί να προσθέσει το συγκεκριμένο sku στη λίστα με τα προϊόντα του. Αν κάποιο sku είναι ήδη στη λίστα του χρήστη τότε εμφανίζεται κουμπί που του επιτρέπει να το αφαιρέσει όπως φαίνεται στην εικόνα 5.3. Επίσης για κάθε sku υπάρχει και μια εικόνα. Αυτή η εικόνα δεν είναι αποθηκευμένη στην εφαρμογή αλλά εμφανίζεται από το skroutz.

Σχήμα 5.3: Λίστα προϊόντων. Μόνο για τους εγγεγραμμένους χρήστες κάτω από κάθε προϊόν υπάρχει η επιλογή για την αφαίρεσή του ή την προσθήκη του στη λίστα προϊόντων του χρήστη ανάλογα με το αν είναι ή δεν είναι ήδη σε αυτή.

Για να υποστηριχθεί αυτή η λειτουργία με το που ο χρήστης επισκεφθεί τη σελίδα εμφάνισης

των προϊόντων η εφαρμογή ανασύρει όλα τα προϊόντα από τη βάση δεδομένων και τα εμφανίζει στο χρήστη όπως φαίνεται στο διάγραμμα 5.4.



Σχήμα 5.4: Διάγραμμα εμφάνισης προϊόντων. Ο χρήστης ζητάει την κεντρική σελίδα εμφάνισης skus και προϊόντων. Τα δεδομένα προϊόντων και καταστημάτων αντλούνται από τη βάση δεδομένων και παρουσιάζονται στο χρήστη. Στην σελίδα υπάρχει επίσης και μια περιοχή που περιέχει τα προϊόντα που έχει προσθέσει ο χρήστης στη λίστα του.

5.3 Αλληλεπίδραση χρήστη με τη λίστα προϊόντων του

Οι εγγεγραμμένοι χρήστες αφού συνδεθούν έχουν τη δυνατότητα να προσθέσουν και να αφαιρέσουν προϊόντα από τη λίστα τους όπως φαίνεται στο σχήμα 5.6. Η λίστα του χρήστη που φαίνεται στην εικόνα 5.5 διαθέτει μια μικρογραφία των προϊόντων που έχει ο χρήστης στη λίστα τους μαζί με ένα κουμπί που του επιτρέπει να το αφαιρέσει. Για να υποστηριχθεί αυτή η λειτουργία η εφαρμογή διαθέτει έναν ξεχωριστό πίνακα user_skus όπου αποθηκεύονται τα προϊόντα των χρηστών. Η δομή του πίνακα περιγράφεται στον πίνακα 4.5.

Σχήμα 5.5: Λίστα προϊόντων χρήστη. Δίπλα από κάθε προϊόν υπάρχει η επιλογή για την αφαίρεσή του, ενώ στο τέλος υπάρχει σύνδεσμος όπου οδηγεί στην επισκόπηση των προϊόντων.

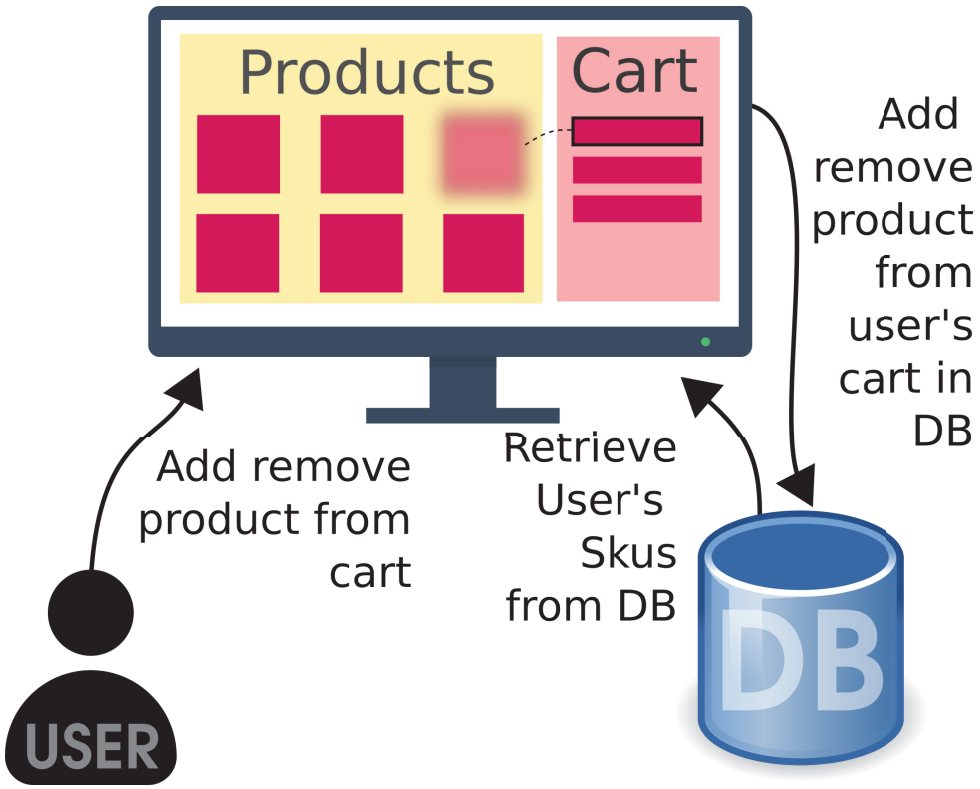
5.4 Επισκόπηση προϊόντων του χρήστη

Κάτω από τη λίστα προϊόντων του χρήστη όπως φαίνεται και στην εικόνα 5.5 υπάρχει η επιλογή για επισκόπηση των προϊόντων του. Πατώντας αυτή την επιλογή οδηγείται στη σελίδα επισκόπησης των προϊόντων του όπου του εμφανίζονται πιθανοί συνδυασμοί καταστημάτων όπου μπορεί να αγοράσει τα προϊόντα που έχει στη λίστα του. Προσφέρονται τρεις επιλογές:

- Διαθεσιμότητα
  Εδώ εμφανίζονται τα καταστήματα όπου διαθέτουν τα προϊόντα πρωτίστως σε άμεση διαθεσιμότητα και δευτερευόντως σε χαμηλότερη τιμή Στην εικόνα 5.8 φαίνεται ένα παράδειγμα εμφάνισης του αλγορίθμου.

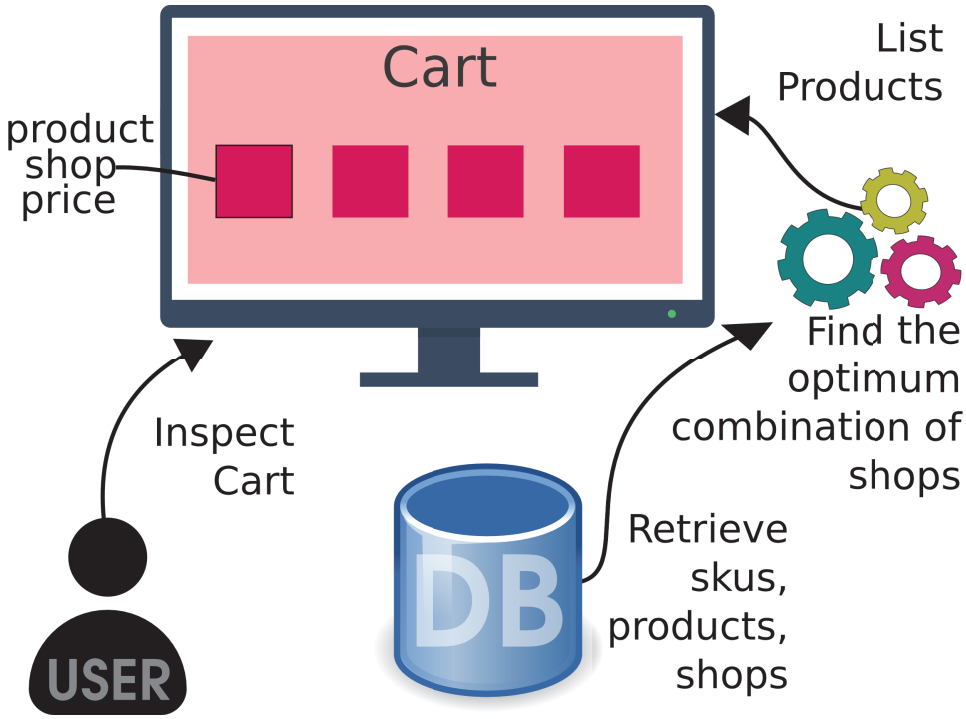
- Φθηνότερα
  Εδώ εμφανίζονται τα καταστήματα όπου διαθέτουν το προϊόν στη χαμηλότερη δυνατή τιμή. Στην εικόνα 5.9 φαίνεται ένα παράδειγμα εμφάνισης του αλγορίθμου.

- Μικρότερος αριθμός καταστημάτων
  Εδώ ο αντίστοιχος αλγόριθμος επιλογής καταστημάτων προσπαθεί να επιλέξει τα προϊόντα με τέτοιο τρόπο ώστε να ελαχιστοποιηθεί ο αριθμός διαφορετικών καταστημάτων που προτείνονται στο χρήστη. Στην εικόνα 5.10 φαίνεται ένα παράδειγμα εμφάνισης του αλγορίθμου.



Σχήμα 5.6: Διάγραμμα αλληλεπίδρασης χρήστη με τη λίστα προϊόντων του. Ο χρήστης βρίσκεται στην κεντρική σελίδα εμφάνισης skus και προϊόντων. Δίπλα από κάθε προϊόν υπάρχει ένα κουμπί που επιτρέπει στο χρήστη να το βάλει στην προσωπική του λίστα με προϊόντα. Επίσης στα προϊόντα που βρίσκονται ήδη στην προσωπική του λίστα, υπάρχει μια επιλογή για να τα αφαιρέσει από τη λίστα. Όταν ο χρήστης αλληλεπιδρά με το καλάθι του ενημερώνεται η βάση δεδομένων.

Για να υποστηριχθεί από την εφαρμογή αυτή η λειτουργία, όταν ο χρήστης επισκέπτεται την σελίδα επισκόπησης των προϊόντων του, η εφαρμογή ανασύρει από τη βάση δεδομένων τα δεδομένα που σχετίζονται με τα skus που έχει ο χρήστης στη λίστα του και εφαρμόζει τους τρεις αλγορίθμους επιλογής καταστημάτων που περιγράφονται παραπάνω. Η διαδικασία αυτή απεικονίζεται στο διάγραμμα 5.7.



Σχήμα 5.7: Διάγραμμα επισκόπησης προϊόντων του χρήστη. Όταν ο χρήστης επισκέπτεται τη σελίδα επισκόπησης των προϊόντων του, τα δεδομένα προϊόντων και καταστημάτων αντλούνται από τη βάση δεδομένων και η εφαρμογή εκτελεί τον αλγόριθμο βέλτιστου συνδυασμού προϊόντων / καταστημάτων / τιμών με κριτήριο το κόστος ή το χρόνο αποστολής. Ο συνδυασμός προϊόντων που επέλεξε ο χρήστης παρουσιάζονται στο χρήστη.

Κατάστημα	Προϊόν	Διαθεσιμότητα	Τιμή
Lenshop lenshop.gr ελάχιστα μεταφορικά 3.3€ δωρεάν μεταφορικά από 50.0€	Woody's Barchelona Hiroto 0,33 	Σε απόθεμα	112.00€
	Porsche Design 8478-C 	Σε απόθεμα	269.00€
		σύνολο:	**381.00€**
Oramaoptics ORAMA ελάχιστα μεταφορικά 4.0€ δωρεάν μεταφορικά από 50.0€	Arnette 4177 22074V 59 	Σε απόθεμα	69.60€
		σύνολο:	**69.60€**

Το κόστος είναι: 450.60€

Σχήμα 5.8: Παράδειγμα εμφάνισης καταστημάτων με βάση τη διαθεσιμότητά τους. Παρατηρούμε ότι όλα τα προϊόντα είναι άμεσα διαθέσιμα αντίθετα από τις άλλες επιλογές καταστημάτων.

Κατάστημα	Προϊόν	Διαθεσιμότητα	Τιμή
Lenshop lenshop.gr ελάχιστα μεταφορικά 3.3€ δωρεάν μεταφορικά από 50.0€	Woody's Barchelona Hiroto 0,33 	Σε απόθεμα	112.00€
	Porsche Design 8478-C 	Σε απόθεμα	269.00€
		σύνολο:	**381.00€**
Sunoptic sunoptic ελάχιστα μεταφορικά 5.0€ δωρεάν μεταφορικά από 50.0€	ARNETTE 4177 2207/4V 59 WITCH DOCTOR 	Κατόπιν Παραγγελίας	64.90€
		σύνολο:	**64.90€**

Το κόστος είναι: 445.90€

Σχήμα 5.9: Παράδειγμα εμφάνισης καταστημάτων με βάση τη τιμή τους. Παρατηρούμε ότι και τα δύο προϊόντα είναι φθηνότερα από τις άλλες επιλογές καταστημάτων.

Κατάστημα	Προϊόν	Διαθεσιμότητα	Τιμή
Fasma Optica φασμα ελάχιστα μεταφορικά 5.0€ δωρεάν μεταφορικά από 50.0€	WOODYS BARCELONA HIROTO/0.37/4826 	Σε απόθεμα	134.00€
	PORSCHE DESIGN 8478N/A/6910 	Σε απόθεμα	338.00€
	ARNETTE 4177/22074V/59 	Σε απόθεμα	77.00€
		σύνολο:	**549.00€**

Το κόστος είναι: 549.00€

Σχήμα 5.10: Παράδειγμα εμφάνισης καταστημάτων με βάση τα καταστήματα. Παρατηρούμε ότι όλα τα προϊόντα προέρχονται από το ίδιο κατάστημα.

Κεφάλαιο 6

Αλγόριθμος επιλογής καταστημάτων

6.1 Παράδειγμα χρήσης των αλγορίθμων

Έστω ότι η μεταβλητή `user` περιέχει ένα χρήστη της εφαρμογής ο οποίος έχει δύο προϊόντα στην λίστα προϊόντων του τότε η κλήση της συνάρτησης `cheapest_products` επιστρέφει ένα `array` με τα φθηνότερα προϊόντα.

```
# Κλήση συνάρτησης
user.cheapest_products()

# Array που επιστρέφεται. Το array έχει δύο προϊόντα
[
    # Πρώτο προϊόν
    {
                  :id => 277,
                :name => "RayBan RB9506S 200/90 50 METAL",
              :sku_id => 19,
             :shop_id => 76,
        :availability => "Κατόπιν Παραγγελίας",
               :price => 50.0,
          :created_at => Thu, 02 Apr 2015 10:39:39 UTC +00:00,
          :updated_at => Thu, 02 Apr 2015 10:39:39 UTC +00:00
    },

    # Δεύτερο προϊόν
```

```
    {
                  :id => 352,
                :name => "Porsche Design 8478-C",
              :sku_id => 23,
             :shop_id => 118,
        :availability => "Σε απόθεμα",
               :price => 269.0,
          :created_at => Fri, 03 Apr 2015 10:55:47 UTC +00:00,
          :updated_at => Fri, 03 Apr 2015 10:55:47 UTC +00:00
    }
]
```

6.2 Αλγόριθμος επιλογής προϊόντων με βάση τη διαθεσιμότητα

Η παρακάτω συνάρτηση είναι μια συνάρτηση που έχει το μοντέλο `User` της εφαρμογής. Για να υπολογίσει τα προϊόντα χρησιμοποιεί τη βοηθητική συνάρτηση `sku.fastest_product`.

```
# Αλγόριθμος επιλογής προϊόντων με βάση τη διαθεσιμότητα
def fastest_products

  # Αρχικοποίηση μεταβλητής όπου θα αποθηκευθούν τα προϊόντα
  fastest_products = []

  skus.each do |sku|
    # έλεγχος για το αν κάποιο προϊόν δεν υπάρχει
    unless sku.fastest_product.nil?
      # εισαγωγή του προϊόντος με την καλύτερη
      # διαθεσιμότητα στο array fastest_products.
      fastest_products.push(sku.fastest_product)
    end
  end

  # επιστροφή array με τα προϊόντα με τη
  # καλύτερη διαθεσιμότητα
```

```
  return fastest_products
end
```

Η παρακάτω συνάρτηση είναι μια συνάρτηση που διαθέτει το μοντέλο Sku της εφαρμογής. Είναι βοηθητική και καλείται από τη συνάρτηση `user.fastest_products`.

```
# Αλγόριθμος επιλογής προϊόντος με βάση τη διαθεσιμότητά του
def fastest_product

  # Αρχικοποίηση μεταβλητής όπου θα αποθηκευθούν τα προϊόντα
  fastest_products = []

  # Τα δεδομένα που προέρχονται από το skroutz και αφορούν τη
  # διαθεσιμότητα των προϊόντων έχουν τις εξής δυνατές τιμές:
  # • Σε απόθεμα
  # • 1 έως 3 ημέρες
  # • 4 έως 7 ημέρες
  # • 7+ ημερες
  # • Κατόπιν Παραγγελίας
  # • nil (χωρίς τιμή)
  #
  # Αυτές οι τιμές είναι σε μορφή κειμένου και βρίσκονται
  # στη μεταβλητή possible_availabilities
  possible_availabilities = ["Σε απόθεμα",
                             "1 έως 3 ημέρες",
                             "4 έως 7 ημέρες",
                             "7+ ημερες",
                             "Κατόπιν Παραγγελίας",
                             nil]

  # Με μια δομή επανάληψης επιλέγουμε πρώτα τα στοιχεία που
  # έχουν availability "σε απόθεμα". Αυτά τα στοιχεία τα
  # ταξινομούμε με βάση την τιμή τους και τα εισάγουμε στη
  # μεταβλητή cheapest_products μετά κάνουμε το ίδιο για τα
  # στοιχεία που έχουν availability "1 έως 3 ημέρες"
  # και ούτω καθεξής.
```

```ruby
# για κάθε τιμή availability
possible_availabilities.each do |availability|

  # Η συνάρτηση select είναι μια συνάρτηση που έχει η δομή
  # Array της Ruby και χρησιμοποιείται για να φιλτράρουμε
  # τα στοιχεία ενός Array. Στην περίπτωσή μας φιλτράρουμε
  # τα στοιχεία του Array products με βάση την τιμή
  # availability που έχουν. Τα στοιχεία που τηρούν αυτό
  # τον έλεγχο τα αποθηκεύουμε προσωρινά στην μεταβλητή tmp
  tmp = products.select do |p|
    p.availability == availability
  end

  # Η συνάρτηση sort είναι μια συνάρτηση που έχει η δομή
  # Array της Ruby και χρησιμοποιείται για να ταξινομήσουμε
  # τα στοιχεία του array που θέλουμε με βάση κάποια τιμή
  # που έχουν. Στην περίπτωσή μας ταξινομούμε τα στοιχεία
  # του Array tmp με βάση την τιμή που έχουν. Το
  # ταξινομημένο array αποθηκεύεται πάλι στη μεταβλητή tmp
  tmp.sort! do |x,y|
    y[:price] <=> x[:price]
  end

  # αντιστροφή του array tmp ώστε τα φθηνότερα
  # προϊόντα να πάνε πρώτα
  tmp.reverse!

  # Τέλος το array tmp συνενώνεται στο τέλος του
  # array fastest_products
  fastest_products += tmp
end

# Η συνάρτηση επιστρέφει το πρώτο προϊόν από το array
# fastest_products που περιέχει τα προϊόντα που πρωτίστως
# βρίσκονται σε διαθεσιμότητα και δευτερευόντως
# έχουν χαμηλότερη τιμή.
return fastest_products.first
```

```
end
```

6.3 Αλγόριθμος επιλογής προϊόντων με βάση τη τιμή

Η παρακάτω συνάρτηση είναι μια συνάρτηση που διαθέτει το μοντέλο `User` της εφαρμογής. Για να υπολογίσει τα προϊόντα χρησιμοποιεί τη βοηθητική συνάρτηση `sku.cheapest_product`.

```
# Αλγόριθμος επιλογής προϊόντων με βάση τη τιμή
def cheapest_products

  # Αρχικοποίηση μεταβλητής όπου θα αποθηκευθούν τα προϊόντα
  cheapest_products = []

  skus.each do |sku|
    # έλεγχος για το αν κάποιο προίόν δεν υπάρχει
    unless sku.cheapest_product.nil?
      # εισαγωγή του προϊόντος με την καλύτερη τιμή
      # στο array cheapest_products.
      cheapest_products.push(sku.cheapest_product)
    end
  end

  # επιστροφή array με τα προϊόντα με τη καλύτερη τιμή
  return cheapest_products
end
```

Η παρακάτω συνάρτηση είναι βοηθητική, τη διαθέτει το μοντέλο `Sku` και καλείται από τη συνάρτηση `user.cheapest_products`.

```
# Αλγόριθμος επιλογής φθηνότερου προϊόντος
def cheapest_product

  # ταξινόμηση των προϊόντων με βάση την τιμή τους και
  # ανάθεση του ταξινομημένου array στην μεταβλητή tmp
  tmp = products.sort do |x,y|
```

```
    y[:price] <=> x[:price]
  end

  # το τελευταίο στοιχείο του array είναι και το φθηνότερο
  # οπότε επέστρεψε αυτό
  return tmp.last
end
```

6.4 Αλγόριθμος επιλογής προϊόντων με βάση τα καταστήματα

Η παρακάτω συνάρτηση είναι μια συνάρτηση που έχει το μοντέλο `User` της εφαρμογής. Για να υπολογίσει τα προϊόντα χρησιμοποιεί τη βοηθητική συνάρτηση `sku.product_by_shop`

```
# Αλγόριθμος επιλογής προϊόντων με βάση τα καταστήματα
def common_shop_products

  # Αρχικοποίηση μεταβλητής όπου θα αποθηκευθούν τα προϊόντα
  products_list = []

  # Αρχικοποίηση βοηθητικής μεταβλητής μετρητή καταστημάτων
  shops_count = Hash.new(0)

  # Επαναληπτική δομή που δημιουργει μια κατάσταση με τα
  # καταστήματα στα οποία περιέχεται ο μεγαλύτερος αριθμός
  # προϊόντων που έχει βάλει ο χρήστης στη λίστα του
  skus.each do |sku|
    sku.shops.uniq.each do |shop|
      shops_count[shop] += 1
    end
  end

  # Ταξινόμηση της παραπάνω κατάστασης.
  # Τα καταστήματα με τα περισσότερα προϊόντα
  # βρίσκονται στην αρχή του array
```

```
  shops_list = shops_count.to_a.sort do |x,y|
    y[1] <=> x[1]
  end

  # Από το array εξάγουμε τη στήλη με τα ονόματα των
  # καταστημάτων οπότε τώρα το array shops_list περιέχει
  # μια ταξινομημένη λίστα με καταστήματα
  shops_list.collect!{|x| x[0]}

  skus.each do |sku|

    # η μεταβλητή product περιέχει ένα προϊόν του οποίου
    # το κατάστημα βρίσκεται υψηλότερα στη λίστα
    # καταστημάτων shops_list
    product= sku.product_by_shop(shops_list)

    # έλεγχος για το αν κάποιο προϊόν δεν υπάρχει
    unless product.nil?
      # εισαγωγή του προϊόντος στο array products_list.
      products_list.push(product)
    end
  end

  # επιστροφή array με τα προϊόντα
  return products_list
end
```

Η παρακάτω συνάρτηση είναι βοηθητική, τη διαθέτει το μοντέλο Sku και καλείται από τη συνάρτηση `user.common_shop_products`

```
# Αλγόριθμος επιλογής προϊόντος με βάση το κατάστημα
#
# ο αλγόριθμος δέχεται σαν όρισμα μια ταξινομημένη λίστα
# με τα καταστήματα από τα οποία θα προτιμούσαμε να αγοράσουμε
# το προϊόν. Και επιστρέφει το προϊόν που βρίσκεται στο
# κατάστημα υψηλότερης προτίμησης
def product_by_shop (shops_list)
```

```
  shops_list.each do |shop|
    products.each do |product|
      if product.shop == shop
        return product
      end
    end
  end
  return nil
end
```

Κεφάλαιο 7

Χρήση υπηρεσιών Υπολογιστικού Νέφους

7.1 Heroku.com

Για να είναι η πτυχιακή μας πιο ολοκληρωμένη, αποφασίσαμε να εγκαταστήσουμε την εφαρμογή μας σε μια υπηρεσία Υπολογιστικού Νέφους.

Μετά από αναζήτηση πιθανών υπηρεσιών που ενδείκνυται για εφαρμογές Ruby on Rails, αποφασίσαμε να χρησιμοποιήσουμε την υπηρεσία Heroku [3]. Το Heroku ανοίκει σε μια υποκατηγορία υπηρεσιών Υπολογιστικού Νέφους που ονομάζεται PaaS (Platform as a Service) [18].

Μετά από εγγραφή στην υπηρεσία heroku διαπιστώσαμε ότι έχει εύχρηστο περιβάλλον και αναλυτικές οδηγίες για το πως μπορεί να ανεβάσει κάποιος μια εφαρμογή Ruby on Rails στην υπηρεσία.

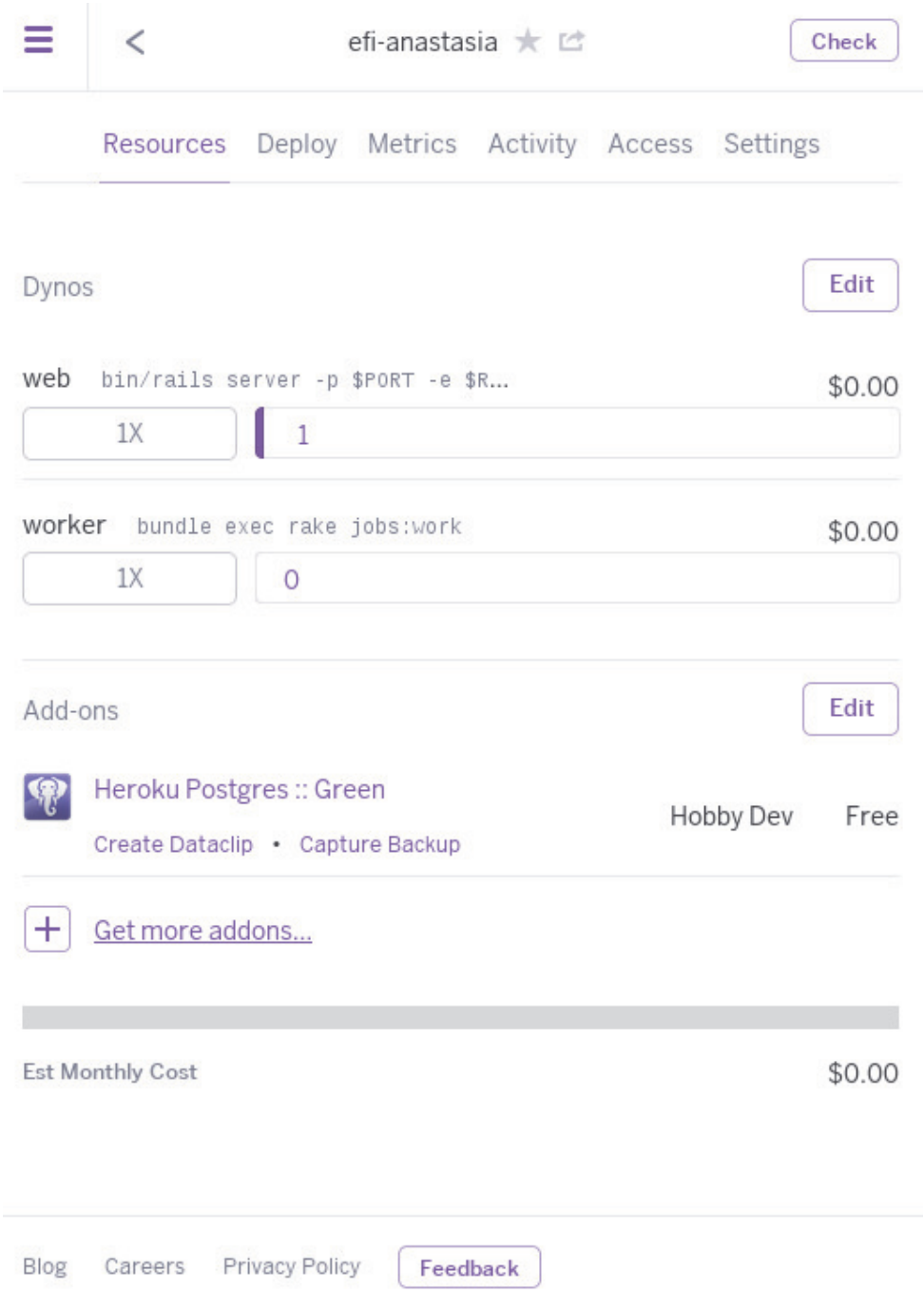
Το περιβάλλον διαχείρισης μιας εφαρμογής στο heroku φαίνεται στην εικόνα 7.1. Διαθέτει επιλογές για να αναβαθμίσει κάποιος το δωρεάν πλάνο υπηρεσιών και να πάει σε πλάνο επί πληρωμή και πολλές άλλες επιλογές διαχείρισης της εφαρμογής. Εμείς ονομάσαμε την εφαρμογή `efi-anastasia` και είναι διαθέσιμη στον ιστότοπο
http://efi-anastasia.herokuapp.com

7.2 Gihub.com

Για να κάνουμε τον πηγαίο κώδικα της εφαρμογής μας διαθέσιμο μέσω του διαδικτύου, επιλέξαμε το github [2] που είναι ένα online σύστημα διαχείρησης εκδόσεων λογισμικού. Η διαδικασία για να γίνει αυτό είναι αρκετά απλή και περιλαμβάνει τα εξής βήματα.

1. Δημιουργία λογαριασμού χρήστη στο github
2. Δημιουργία ενός κενού αποθετηρίου κώδικα μέσω του διαδικτυακού περιβάλλοντος του github
3. Συγχρονισμός του απομακρυσμένου (κενού) αποθετηρίου με ένα τοπικό φάκελο κάνοντας χρήση της εντολής `git clone`
4. Εισαγωγή των αρχείων της εφαρμογής στον κενό τοπικό φάκελο.
5. Εισαγωγή των αρχείων της εφαρμογής στο τοπικό αποθετήριο κάνοντας χρήση της εντολής `git add -A .` και της εντολής `git commit`
6. Ενημέρωση του απομακρυσμένου αποθετηρίου μέ την εντολή `git push`

Το αποθετήριο κώδικα βρίσκεται στη διεύθυνση https://github.com/efist/efi-anastasia



Σχήμα 7.1: Περιβάλλον διαχείρισης της εφαρμογής μας στο Heroku. Από εδώ μπορούμε εύκολα να χειριστούμε διάφορες παραμέτρους της εφαρμογής και να αυξήσουμε τους πόρους που έχει διαθέσει το heroku στην εφαρμογή μας (αυξάνοντας παράλληλα και το κόστος που τώρα είναι δωρεάν).

Κεφάλαιο 8

Επίλογος

Κατά τη διάρκεια ανάπτυξης της εφαρμογής αντιμετωπίσαμε πολλά διλήμματα σχετικά με πλήθος σχεδιαστικών αποφάσεων γύρω από τη μορφή, τεχνικές προδιαγραφές και χαρακτηριστικά που θα θέλαμε να έχει.

Ξεκινώντας από το από ποια από τις υπάρχουσες εφαρμογές και αντίστοιχα API θα χρησιμοποιούσαμε επιλέξαμε το API του skroutz λόγω της απλότητας του και γιατί μας εξυπηρετούσε για το σκοπό που το θέλαμε. Επίσης θεωρήσαμε θετικό ότι τα προϊόντα που μας διαθέτει το skroutz μέσω του API του έχουν ονόματα και περιγραφές στην ελληνική γλώσσα. Επίσης θετικό ήταν ότι κατά την επικοινωνία που είχαμε με τους προγραμματιστές του skroutz ώστε να ζητήσουμε πρόσβαση στο API τους ήταν φιλικοί και πρόθυμοι να μας βοηθήσουν σε τυχόν προβλήματα που θα αντιμετωπίζαμε.

Συνεχίζοντας επιλέγοντας περιβάλλον ανάπτυξης εφαρμογών διαδικτύου ήμασταν ανάμεσα σε Java EE, στο Django σε Python στο Ruby on Rails σε Ruby και σε PHP. Κατα τη συγγραφή του πρώτου κεφαλαίου της εργασίας διαπιστώσαμε ότι πολλές μεγάλες διαδικτυακές εφαρμογές ηλεκτρονικού εμπορίου χρησιμοποιούν το Ruby on Rails και επιπροσθέτως το skroutz το χρησιμοποιεί και αυτό. Αποφασίσαμε λοιπόν να το χρησιμοποιήσουμε και εμείς για την ανάπτυξη του δικού μας πράκτορα σύγκρισης αγορών.

Στη συνέχεια αντιμετωπίσαμε πρόβλημα κατά την ανάπτυξη ενός αλγορίθμου επιλογής καταστημάτων τέτοιο ώστε να είναι βέλτιστος και συνυπολογίζει στην επιλογή των καταστημάτων το γεγονός ότι αν για παράδειγμα επιλεγούν δύο προϊόντα από το ίδιο κατάστημα στο τελικό κόστος θα πρέπει να υπολογίσουμε λιγότερα μεταφορικά έξοδα. Ψάχνοντας περισσότερο διαπιστώσαμε ότι τέτοιος αλγόριθμος είναι τεχνικά και υπολογιστικά δύσκολο να φτιαχτεί. Επίσης παρατηρήσαμε ότι το Amazon το ebay και το skroutz δεν έχουν ακόμα φτιάξει τέτοιο αλγόριθμο. Σε επικοινωνία που είχαμε με το skroutz ενημερωθήκαμε ότι προσπαθούν να

αναπτύξουν ένα τέτοιο αλγόριθμο. Αν σκεφτούμε τις παραμέτρους που θα πρέπει να υπολογίζει ένας τέτοιος αλγόριθμος η εικόνα σχετικά με την μεγάλη πολυπλοκότητά του ξεκαθαρίζει ακόμα περισσότερο. Συγκεκριμένα κάθε κατάστημα έχει διαφορετική πολιτική για τα μεταφορικά δηλαδή διαφορετικές χρεώσεις ανάλογα με την απόσταση από το μαγαζί, το συνολικό κόστος της παραγγελίας και ακόμα και για το συνολικό βάρος της παραγγελίας. Και προφανώς θα πρέπει να είναι αρκετά γρήγορος ώστε με το που ζητήσει ο χρήστης τη σελίδα με τα αποτελέσματα του αλγορίθμου, αυτά να υπολογιστούν άμεσα και να εμφανιστούν στο χρήστη. Στη δική μας υλοποίηση ακολουθήσαμε μια πιο απλουστευμένη προσέγγιση όπου για ένα δεδομένο σύνολο προϊόντων που έχει επιλέξει ο χρήστης προσπαθούμε αυτά να προέρχονται από τα ίδια καταστήματα. Δηλαδή για ένα δεδομένο σύνολο προϊόντων που έχει επιλέξει ο χρήστης να χρειαστεί να κάνει λιγότερες παραγγελίες για να τα αποκτήσει. Αυτό εξυπηρετεί τους χρήστες της εφαρμογής γιατί μπορούν να συγκρίνουν άμεσα τα αποτελέσματα αυτού το αλγορίθμου με τα αποτελέσματα του αλγορίθμου φθηνότερων προϊόντων.

Επίσης θεωρήσαμε σημαντικό να ανεβάσουμε την εφαρμογή online σε μια δωρεαν υπηρεσία PaaS (Platform as a Service). Δημοφιλή ανάμεσα στους προγραμματιστές του περιβάλλοντος Ruby on Rails είναι η υπηρεσία heroku οπότε επιλέξαμε να χρησιμοποιήσουμε αυτή. Διαπιστώσαμε με ευχαρίστηση ότι για το ανέβασμα μιας εφαρμογής διαθέτει αναλυτικές οδηγίες βήμα προς βήμα που βοηθάνε υπερβολικά τη διαδικασία. Η εφαρμογή που δημιουργήσαμε ονομάζεται efi-anastasia και είναι προσβάσιμη από τη διεύθυνση http://efi-anastasia.herokuapp.com

Τέλος ανεβάσαμε τον κώδικα της πτυχιακής στο Github, ένα online σύστημα διαχείρησης εκδόσεων λογισμικού. Το αποθετήριο κώδικα στο οποίο είναι διαθέσιμη η πτυχιακή μας βρίσκεται στη διεύθυνση https://github.com/efist/efi-anastasia

Βιβλιογραφία

[1] Amazon.com. Amazon affiliate program.
https://affiliate-program.amazon.com/gp/advertising/api/detail/main.html,
21 Ιουνίου 2014. 16

[2] Github. Github.
https://github.com,
30 Μαίου 2015. 54

[3] Heroku. Heroku.
http://heroku.com,
30 Μαίου 2015. 54

[4] U.S. Department of Transportation vectorization. Relational model.
http://commons.wikimedia.org/wiki/File:Relational_Model.svg,
5 Οκτωβρίου 2014. 26

[5] Ruby on Rails API. Secure password.
http://api.rubyonrails.org/classes/ActiveModel/SecurePassword/ClassMethods.html,
24 Φεβρουαρίου 2015. 34

[6] ProgrammableWeb.com. Etsy.com api.
https://affiliate-program.amazon.com/gp/advertising/api/detail/main.html,
21 Ιουνίου 2014. 19

[7] ProgrammableWeb.com. Groupon.com api.
https://affiliate-program.amazon.com/gp/advertising/api/detail/main.html,
21 Ιουνίου 2014. 19

[8] ProgrammableWeb.com. Shopify.com api.
https://affiliate-program.amazon.com/gp/advertising/api/detail/main.html,
21 Ιουνίου 2014. 20

[9] ProgrammableWeb.com. Shopping.com api.
https://affiliate-program.amazon.com/gp/advertising/api/detail/main.html,
21 Ιουνίου 2014. 19

[10] ProgrammableWeb.com. Shopzilla.com api.
https://affiliate-program.amazon.com/gp/advertising/api/detail/main.html,
21 Ιουνίου 2014. 19

[11] RailsGuides. Active record basics.
http://guides.rubyonrails.org/active_record_basics.html,
12 Οκτωβρίου 2014. 27, 29, 30

[12] RailsGuides. Active record migrations.
http://guides.rubyonrails.org/migrations.html,
12 Οκτωβρίου 2014. 29

[13] RailsGuides. Active record query interface.
http://guides.rubyonrails.org/active_record_querying.html,
12 Οκτωβρίου 2014. 30

[14] Skroutz.gr. Authentication.
http://docs.skroutz.gr/apiv3/guides/authentication,
21 Ιουνίου 2014. 12

[15] stackoverflow. What is an object-relational mapping framework?
http://stackoverflow.com/questions/1152299/what-is-an-object-relational-mapping-framework,
12 Οκτωβρίου 2014. 27, 28

[16] Wikipedia. Comma-separated values.
http://en.wikipedia.org/wiki/Comma-separated_values,
2 Οκτωβρίου 2014. 23

[17] Wikipedia. Xml.
http://en.wikipedia.org/wiki/XML,
2 Οκτωβρίου 2014. 23

[18] Wikipedia. Platform as a service.
https://en.wikipedia.org/wiki/Platform_as_a_service,
20 Μαρτίου 2015. 54

[19] Wikipedia. Amazon product advertising api.
https://en.wikipedia.org/wiki/Amazon_Product_Advertising_API,
21 Ιουνίου 2014. 16

[20] Wikipedia. Amazon.com.
https://el.wikipedia.org/wiki/Amazon.com,
21 Ιουνίου 2014. 16

[21] Wikipedia. bcrypt.
https://en.wikipedia.org/wiki/Bcrypt,
24 Φεβρουαρίου 2015. 34

[22] Wikipedia. Database management systems.
http://en.wikipedia.org/wiki/Database,
5 Οκτωβρίου 2014. 25

[23] Wikipedia. Json.
http://en.wikipedia.org/wiki/JSON,
5 Οκτωβρίου 2014. 24

[24] Βικιπαίδεια. Αντικειμενοστρεφής προγραμματισμός.
http://el.wikipedia.org/wiki/Αντικειμενοστρεφής_προγραμματισμός,
12 Οκτωβρίου 2014. 27